Formal Verification of Conflict Detection Algorithms for Arbitrary Trajectories*

Anthony Narkawicz
NASA Langley Research Center, Hampton, VA 23681, USA
`Anthony.Narkawicz@nasa.gov`

César A. Muñoz
NASA Langley Research Center, Hampton, VA 23681, USA
`Cesar.A.Munoz@nasa.gov`

Abstract

This paper presents an approach for developing formally verifiable conflict detection algorithms for aircraft flying arbitrary, nonlinear trajectories. The approach uses a multivariate polynomial global optimization algorithm based on Bernstein polynomials. Since any continuous function on a closed interval, such as an aircraft trajectory within a closed interval of time, can be uniformly approximated by a Bernstein polynomial, this global optimization algorithm can be used to define conflict detection algorithms for arbitrarily complicated trajectories. Conflict detection algorithms developed using this approach can be formally verified in a mechanical theorem prover. This represents an improvement over standard approaches to conflict detection for complex trajectories that essentially search for conflicts by testing many future states and are therefore not guaranteed to detect a given conflict. The proposed approach is illustrated with a formally verified conflict detection algorithm.

Keywords: aircraft conflict detection, formal verification, global optimization, theorem proving
AMS subject classifications: 65G20, 03B35, 68T15

1 Introduction

In air traffic management, a loss of separation is a violation of the separation requirement between two aircraft. This separation requirement is given by a minimum horizontal separation, e.g., 5 nautical miles, and a minimum vertical separation, e.g., 1000 feet [14]. Assuming an aircraft trajectory model, conflict detection algorithms predict whether or not two aircraft will lose separation within some lookahead time,

*Submitted: August 8, 2011; Revised: August 8, 2011; Accepted: August 8, 2011.

which is typically 5 minutes. When a conflict is detected, conflict resolution algorithms compute resolution maneuvers for the aircraft that maintain the required aircraft separation. Conflict detection and resolution (CD&R) systems are part of computer-based systems that assist pilots and air traffic controllers to maintain safety in the airspace by keeping aircraft separated. These separation assurance systems are critical elements of air/ground distributed operational concepts for the next generation of air traffic management systems such as the US's Next Generation of Air Traffic Systems (NGATS) [24] and Europe's Single European Sky ATM Research (SESAR).[1]

The internal logic of CD&R algorithms relies on the reported current state information of the aircraft, typically 3D position and velocity vectors, and an aircraft trajectory model that propagates the current state information for a given lookahead time. Several state propagation methods for CD&R systems have been proposed [8]. For example, state-based conflict detection algorithms use a linear projection of the current state of the aircraft. This simple aircraft trajectory model corresponds to a point mass moving along a straight line at constant speed. More sophisticated state propagation methods assume nonlinear trajectories or probabilistic trajectory models.

This paper concerns *formal verification* of conflict detection algorithms that model aircraft trajectories as continuous functions over real numbers. In the context of this paper, formal verification refers to computer-checked mathematical proofs that a given algorithm satisfies some safety properties, where the algorithm and the properties are expressed in a formal mathematical notation. In this sense, an algorithm is not a computer program, but a mathematical abstraction of a computer program. Formal verification, as used in this paper, guarantees that algorithms used in critical systems, such as CD&R, are functionally and logically correct, i.e., that assuming an ideal computer platform, the algorithms implement their intended functionalities. A complementary area in computer science known as *software verification* concerns the correctness of algorithm implementations in specific computer platforms. Software verification is out of the scope of this paper. In particular, this paper assumes that the arithmetic used in the description of conflict detection algorithms has *real number* semantics, as opposed to *floating-point number* semantics.

A safety property of conflict detection algorithms known as *soundness* states that all potential conflicts, according to a given aircraft trajectory model, are detected. That is, if in every situation where two aircraft are in conflict, a given algorithm returns true, then that algorithm is sound. For example, an algorithm that always returns true is trivially sound, although it is not a particularly useful one. Another safety property, known as *completeness*, states that the algorithm only detects potential conflicts. That is, if in every situation where an algorithm returns true, the aircraft are actually in conflict, then the algorithm is complete. For example, an algorithm that never returns true is trivially complete, but it is not sound. The notions of soundness and completeness are related to the notions of missed alerts and false alerts, respectively. A conflict detection algorithm that misses alerts negatively affects the safety case for separation assurance systems, since it may indicate that certain maneuvers are safe when they are not. False alerts also have safety implications as they may diminish the confidence that pilots and air traffic controllers have on these systems.

Properties such as soundness and completeness have been formally verified for CD&R algorithms that assume linear trajectories using mechanical theorem provers [10, 13]. A conflict resolution algorithm for curved trajectories has been formally verified using hybrid-model checking techniques [19]. Other type of trajectories, such as piece-

[1]`http://www.eurocontrol.int/content/sesar-and-research.`

wise linear trajectories admit analytical methods [7, 6] and thus, formal verification of these algorithms is feasible. Most CD&R algorithms that handle complicated trajectories either iterate an analytical method at specified discrete steps [2, 17] or rely on numerical approximation methods [21, 18, 1]. Formal verification of these kinds of algorithms is very difficult. In general, soundness and completeness of iterative algorithms cannot be inferred from soundness and completeness of the analytical methods that they iterate since some conflicts may be missed or not correctly solved for input values outside the specified discrete steps. Furthermore, CD&R algorithms based on numerical methods are neither sound nor complete unless the computation errors are correctly accounted for.

This paper presents a numerical approximation approach for designing provably sound and complete conflict detection algorithms for arbitrary aircraft trajectories. In this approach, conflict detection for arbitrary aircraft trajectories is expressed as a global optimization problem and then analytically solved using *Bernstein polynomials* [9]. Methods for global optimization based on Bernstein polynomials allow for the computation of lower and upper bounds for arbitrary continuous functions, e.g., the separation distance between two aircraft, to any desired precision. These bounds are guaranteed to be correct. This is in contrast to some numerical methods for global optimization such as genetic algorithms [5] that do not guarantee the correctness of their results.

The rest of the paper is organized as follows. The formal mathematical notation used in this paper is introduced in Section 2. The conflict detection problem and how this problem can be expressed as a global optimization problem is discussed in Section 3. As part of this work, Bernstein polynomials and their main properties have been formally specified and verified. This formal development of Bernstein polynomials is described in Section 4. A method for finding bounds of the maximum and minimum values of functions that are defined as the maximum of two polynomials is presented in Section 5. That method, which uses Bernstein polynomials, is the core function of a verified conflict detection algorithm for arbitrary trajectories proposed in Section 6. The last section concludes this paper.

The work presented in this paper is part of a research project by the authors that aims at the development of verification technology for global optimization problems.[2] This research project includes the development of a formal library for Bernstein polynomials, which is available as part of the PVS NASA Libraries.[3]

2 Prototype Verification System (PVS)

An important feature of the mathematical development presented in this paper is that it has been specified and formally verified in the proof assistant called Prototype Verification System (PVS) [15]. PVS is a computer program that consists of a *specification language*, i.e., a mathematical notation, and a *theorem prover*, i.e., a symbolic engine that implements the deductive rules of a logic system. The PVS specification language enables the precise definition of mathematical objects such as *functions* and *relations*, and the precise statement of logical formulas such as *lemmas* and *theorems*. Proofs of logical formulas can be mechanically checked using the PVS theorem prover, which guarantees that every proof step is correct and that all possible cases of a proof are

[2]`http://shemesh.larc.nasa.gov/people/cam/Bernstein`.

[3]`http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html`.

covered. All propositions, lemmas and theorems presented in this paper have been mechanically checked in PVS for logical correctness.

The use of a formal language, e.g., in this case the specification language of PVS, enforces rigorous definitions of mathematical objects, where all dependencies are clearly specified. This level of rigor is justified in the development of algorithms for air traffic management given the critical nature of these kinds of systems. In particular, formal verification provides a very high confidence on the correctness of the results presented in this paper. However, this also makes the notation heavy and difficult to read for the non-expert reader. To make this development more accessible to the casual reader, the work presented here uses a simpler mathematical notation that does not require the reader to be familiar with the syntax or semantics of the PVS language.

Despite the use of standard mathematical notation in many places, this paper heavily emphasizes the abstract data structures and specific signatures (types) of the functions used in the algorithms. This includes writing many properties as explicit statements involving function calls. The reason for this is that the intended audience includes not just mathematicians interested in global optimization but also researchers in air traffic management who may want to implement a formally verified conflict detection algorithm whose output can be trusted. Such readers will be interested in specific definitions that can be directly translated to code.

The PVS specification language is *strongly typed*, i.e., all declarations are explicitly typed. This feature guarantees that all PVS functions are total and well-defined. For instance, a mathematical formula that includes a division needs to make explicit the fact that the divisor is different from zero, otherwise the expression would be undefined. In PVS, these conditions are handled by a type system, which is enforced by the PVS type-checker. Since PVS type annotations tend to be verbose, formulas in this paper are implicitly typed, i.e., it is assumed that the type domain of variables is inferred from the context where the formula appears.

PVS is based on higher-order logic, so it supports the definition of functions that return functions or that have functions as arguments. The notation $\lambda x\colon e$ represents an anonymous function that takes as input x and returns e. As discussed above, PVS is a strongly typed language and, therefore, the type of x has to be explicitly declared in PVS. This paper assumes that the type of x is inferred from the context. A function that returns another function is called a *parametric function* and its arguments are called *parameters*. By convention, parameters are sub-indicated, e.g., a parametric function f that given t, x, and y returns a function of type $\mathbb{R} \to \mathbb{R}$ is denoted $f_{t,x,y}$, where t, x, y are the parameters of f. Since $f_{t,x,y}$ has the type $\mathbb{R} \to \mathbb{R}$, the function application $f_{t,x,y}(z)$ has the type $\mathbb{R}$ for any $z \in \mathbb{R}$.

PVS provides numerical types for natural and real numbers that correspond to the mathematical sets $\mathbb{N}$ and $\mathbb{R}$. In PVS, `1/3 + 1/3 + 1/3` is exactly equal to `1` and `sqrt(2)` refers to the unique positive real number such that `sqrt(2)*sqrt`(2) is exactly equal to `2`. The sets of integer and real numbers are unbounded and it is possible to prove properties about arbitrary large or small numbers.

The PVS specification language is a mathematical language rather than a programming language. In PVS, algorithms are defined as mathematical functions. Therefore, functions do not have side effects and variables have the mathematical meaning of arbitrary constant values instead of memory cells as in programming languages. By convention, names of functions that are intended to have a logical meaning, i.e., *predicates*, are written in *italics*. Functions that are intended to be used as algorithmic procedures are written in `typewriter` font.

The PVS specification language supports data structures such as records and tuples. Since records in this paper are mainly used to specify outputs of algorithms, they are also written in `typewriter` font. Record types are defined as $\mathtt{T} = \mathtt{a}_1{:}\mathtt{T}_1 \times \ldots \times \mathtt{a}_n{:}\mathtt{T}_n$, where $\mathtt{T}$ is the name of the record type, $\mathtt{a}_1 \ldots \mathtt{a}_n$ are the names of the fields, and $\mathtt{T}_i$, for $1 \leq i \leq n$, is the type of the field $\mathtt{a}_i$. Field access will be denoted using the dot symbol. The operator `with` is used to overwrite records, e.g., if $\mathtt{t}$ is a record of type $\mathtt{T}$, the record $\mathtt{t}' = \mathtt{t}\ \mathtt{with}\,[\,\mathtt{a}_1 \leftarrow e, \mathtt{a}_n \leftarrow f\,]$ refers to the record that is equal to $\mathtt{t}$ in every field except in $\mathtt{a}_1$ and $\mathtt{a}_n$, where it has the values e and f, respectively, i.e., $\mathtt{t}'.\mathtt{a}_1 = e$, $\mathtt{t}'.\mathtt{a}_n = f$, and for all $1 < i < n$, $\mathtt{t}'.\mathtt{a}_i = \mathtt{t}.\mathtt{a}_i$.

Tuples will be typed in lowercase **boldface**, e.g., $\boldsymbol{a} = (a_0, \ldots, a_{m-1})$ is an *m-tuple* where subindices from 0 to $m-1$ are used to denote particular elements of $\boldsymbol{a}$. Given a positive natural number m, the order $<$ between m-tuples is defined by $\boldsymbol{a} < \boldsymbol{b}$ if and only if $a_j < b_j$ for all $0 \leq j < m$. Similarly, the order $\leq$ between m-tuples is defined by $\boldsymbol{a} \leq \boldsymbol{b}$ if and only if $a_j \leq b_j$ for all $0 \leq j < m$. The m-tuples $\mathbf{0}$ and $\mathbf{1}$ represent the tuples whose components are all 0 and all 1, respectively. A *bounded box* $[\boldsymbol{a}, \boldsymbol{b}]$, where $\boldsymbol{a} < \boldsymbol{b}$, denotes the set of m-tuples greater than or equal to $\boldsymbol{a}$ and less than or equal to $\boldsymbol{b}$. The box $[\mathbf{0}, \mathbf{1}]$ is called the *unit box*.

As in the case of records, tuples can be overwritten using the operator `with`, e.g., if $\boldsymbol{a}$ is an m-tuple, the tuple $\boldsymbol{a}\ \mathtt{with}\,[\,i \leftarrow b\,]$, with $0 \leq i < m$, is equal to $\boldsymbol{a}$ in every index except in i where it has the value b.

Algorithms for conflict detection considered in this paper use a 3D flat earth projection of the airspace. Aircraft positions and velocities in this 3D rectangular coordinate system are represented in PVS as 3-tuples in $\mathbb{R}^3$, where the first and second components denote the horizontal plane and the third component denotes the vertical dimension. Such tuples are appropriately called *vectors*. Components of vectors are sub-indicated by x, y, and z instead of 0, 1, and 2. Furthermore, if $\mathbf{w}$ is the vector (w_x, w_y, w_z), then $\mathbf{w}_{(x,y)}$ denotes the projection of $\mathbf{w}$ in the horizontal plane, i.e., $\mathbf{w}_{(x,y)} = (w_x, w_y)$, and $\mathbf{w}^{\perp}$ denotes the vector $(w_y, -w_x, w_z)$. The notation $\|\mathbf{w}\|$ refers to the norm of the vector $\mathbf{w}$ and the notation $\mathbf{w} \cdot \mathbf{w}'$ refers to the dot product of the vectors $\mathbf{w}$ and $\mathbf{w}'$.

3 Conflict Detection

This section provides a mathematical description of aircraft conflicts for arbitrary trajectories and discusses how the problem of detecting aircraft conflicts can be expressed as a global optimization problem. Since conflicts between multiple aircraft can be detected in a pairwise fashion, only two aircraft are considered. These two aircraft are referred to as the *ownship* and the *intruder*.

The airspace volume is modeled using a flat-earth projection in a 3D rectangular system, i.e., aircraft positions are viewed as points in $\mathbb{R}^3$. In this airspace, the separation requirement between two aircraft is specified as a minimum horizontal separation D and a minimum vertical separation H. Typically, D is 5 nautical miles and H is 1000 feet [14]. In this paper, D and H are considered to be known numerical constants. The separation requirement can be understood as an imaginary horizontal cylinder, called *protected zone*, of height $2H$ and radius D around the intruder aircraft.

A loss of separation between the ownship and the intruder aircraft occurs when the horizontal distance between the aircraft is less than D and the vertical distance is less than H, i.e., when the ownship is in the interior of the intruder's protected zone. Let $\mathbf{s}_o \in \mathbb{R}^3$ and $\mathbf{s}_i \in \mathbb{R}^3$ be the current positions of the ownship and intruder

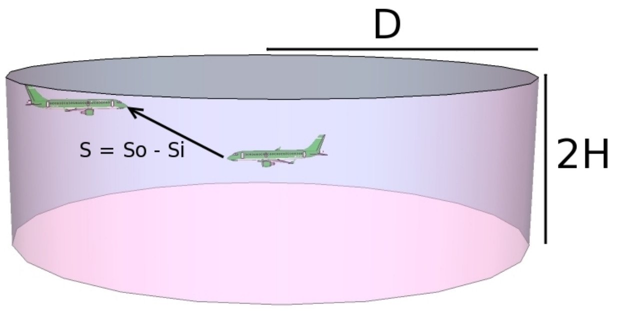

Figure 1: Loss of Separation

aircraft, respectively. Formally, the ownship and intruder aircraft are said to be in *loss of separation* if the following predicate on D, H, $\mathbf{s}_o$, and $\mathbf{s}_i$, holds.

$$\textit{los?}(D, H, \mathbf{s}_o, \mathbf{s}_i) \equiv |s_z| < H \text{ and } \|\mathbf{s}_{(x,y)}\| < D, \text{where } \mathbf{s} = \mathbf{s}_o - \mathbf{s}_i.$$

Loss of separtion is illustrated by Figure 1.

3.1 Trajectories

An aircraft trajectory represents the set of possible positions for the aircraft within a lookahead time T according to some state estimation model [8]. As in the case of D and H, T is assumed to be a known numerical constant.

A state estimation model for CD&R systems may be as simple as a linear projection of the current position at the current constant speed. More complicated models consider uncertainties in the aircraft state due to aircraft dynamics, weather patterns, and other factors. In this paper, an *aircraft trajectory* is a continuous function of type $\mathbb{R}^m \to \mathbb{R}^3$, where the first variable in $\mathbb{R}^m$ represents time and it is bounded by the time interval $[0, T]$. The other variables in $\mathbb{R}^m$ represent uncertainties and are assumed to be bounded as well. Given $\boldsymbol{x} \in \mathbb{R}^m$, the output of a trajectory evaluated at $\boldsymbol{x}$ is a point in $\mathbb{R}^3$ that represents a 3-D position for the aircraft. The following examples give formal definitions of trajectories for several types of state estimation models.

Example 1 (Linear Dynamics Without Uncertainty) *A simple trajectory model for aircraft assumes a linear projection of its current position* $\mathbf{s} \in \mathbb{R}^3$ *along its current velocity* $\mathbf{v} \in \mathbb{R}^3$. *This type of trajectory can be represented by the parametric function* $\texttt{linear}_{\mathbf{s},\mathbf{v}} : \mathbb{R} \to \mathbb{R}^3$*, with parameters* $\mathbf{s}$ *and* $\mathbf{v}$*, defined by*

$$\texttt{linear}_{\mathbf{s},\mathbf{v}}(t) \equiv \mathbf{s} + t\ \mathbf{v}. \tag{1}$$

In this case, $\mathbb{R}^m = \mathbb{R}$*, i.e.,* $m = 1$*, and the variable* $t \in [0, T]$ *represents time. This linear trajectory is illustrated by Figure 2.*

Example 2 (Linear Dynamics With Cross-Track Uncertainty) *In Example 1, the position at time* t *of an aircraft along a linear trajectory at constant speed is given*

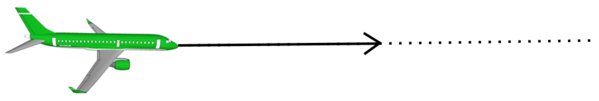

Figure 2: Linear Trajectory

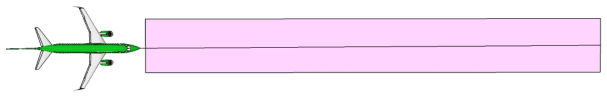

Figure 3: Linear Trajectory with Cross-Track Uncertainty

by $\texttt{linear}_{\mathbf{s},\mathbf{v}}(t)$*, where* $\mathbf{s}$ *and* $\mathbf{v}$ *are the initial position and velocity vector of the aircraft. Uncertainty in the horizontal position of the aircraft is known as* cross-track uncertainty *[7]. If this uncertainty is bounded by some distance* E_v*, then the difference between the actual position and the position only considering cross-track uncertainty at any time is given by* $x\frac{E_v}{\|\mathbf{v}_{(x,y)}\|}\,\mathbf{v}^{\perp}$*, for some* $x \in [-1, 1]$*. This difference vector is perpendicular to the initial velocity of the aircraft. Considering cross-track uncertainty, the trajectory of an aircraft along a linear path can be represented by the parametric function* $\texttt{linearunc}_{\mathbf{s},\mathbf{v},E_v} : \mathbb{R}^3 \to \mathbb{R}^3$*, with parameters* $\mathbf{s}$*,* $\mathbf{v}$*, and* E_v*, defined by*

$$\texttt{linearunc}_{\mathbf{s},\mathbf{v},E_v}(t,x) \equiv \mathbf{s} + t\ \mathbf{v} + x\frac{E_v}{\|\mathbf{v}_{(x,y)}\|}\,\mathbf{v}^{\perp}, \tag{2}$$

where the variable $t \in [0, T]$ *represents time and the variable* $x \in [-1, 1]$ *represents the (often unknown) cross-track uncertainty. This trajectory is illustrated by Figure 3.*

Example 3 (Turn Dynamics Without Uncertainty) *During a steady coordinated turn without friction, the position of an aircraft will follow a circle of radius* $\frac{\nu^2}{g\ \tan\varphi}$*, where* ν *is the true air speed,* g *is the acceleration of gravity, and* φ *is the bank angle of the aircraft. Thus, the trajectory of an aircraft during a turn can be represented by the parametric function* $\texttt{turn}_{\mathbf{s},R,\alpha,\omega,v_z} : \mathbb{R} \to \mathbb{R}$*, with parameters* $\mathbf{s}$*,* R*,* α*,* ω*, and* v_z*, defined by*

$$\texttt{turn}_{\mathbf{s},R,\alpha,\omega,v_z}(t) \equiv \mathbf{s} + (R\sin(\alpha + t\,\omega), R\cos(\alpha + t\,\omega), t\,v_z), \tag{3}$$

where $\mathbf{s}$ *is the center point of the turn,* $\omega = \pm\frac{g}{\nu}\tan\varphi$*,* α *is the angle along the turn at time zero,* $R = \frac{\nu^2}{g\ \tan\varphi}$*, and* v_z *is the vertical speed. The variable* $t \in [0, T]$ *represents time. This trajectory is illustrated by Figure 4.*

Example 4 (Turn Dynamics With Wind and Altitude Uncertainty) *In Example 3, the position at time* t *of an aircraft in a steady turn is given by* $\texttt{turn}_{\nu,g,\varphi,\mathbf{s},\alpha}(t)$*. In the event of a wind where its speed components are varying between* 0 *and* E_w *and assuming that the uncertainty in the altitude of the aircraft is bounded by* E_a*, the*

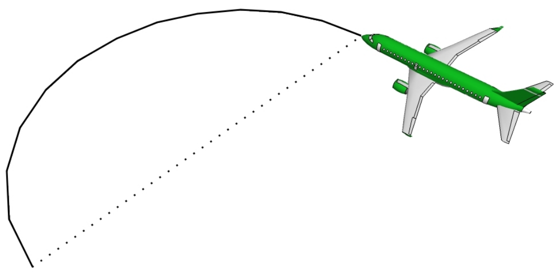

Figure 4: Turning Trajectory for Constant Bank-Angle

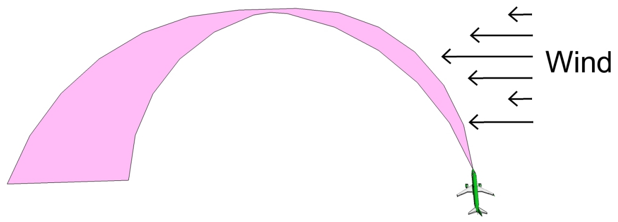

Figure 5: Turning Trajectory with Wind Uncertainty

trajectory of an aircraft during a steady turn can be represented by the parametric function $\textit{\textbf{windturn}}_{\mathbf{s},R,\alpha,\omega,v_z,E_w,E_a} : \mathbb{R}^4 \to \mathbb{R}^3$*, with parameters* $\mathbf{s}$*,* R*,* α*,* ω*,* v_z*,* E_w*, and* E_a*, defined by*

$$\textit{\textbf{windturn}}_{\mathbf{s},R,\alpha,\omega,v_z,E_w,E_a}(t,x,y,z) \equiv \textit{\textbf{turn}}_{\mathbf{s},R,\alpha,\omega,v_z}(t) + (t\,x\,E_w, t\,y\,E_w, z\,E_a), \tag{4}$$

where the variable $t \in [0,T]$ *represents time, the variables* $x, y \in [-1,1]$ *represent wind uncertainty, and the variable* $z \in [-1,1]$ *represents altitude uncertainty. A top-down view of this trajectory is illustrated by Figure 5.*

As illustrated by these examples, the first variable in a trajectory function represents time and is bounded by $[0,T]$. If the trajectory function has more than one variable in the domain, the other variables are bounded in predetermined intervals as well. For instance, in Example 2 the variables x and y, representing along-track and cross-track errors, are bounded in the interval $[-1,1]$.

Trajectories for the ownship and intruder aircraft are denoted by P_o and P_i, respectively. Without loss of generality, it can be assumed that these trajectories have the same variables, that is, they are both functions of type $\mathbb{R}^m \to \mathbb{R}^3$. Indeed, if one trajectory has arguments that are not used by the other trajectory, the other trajectory can be written as a function on its own arguments and those extra arguments as well. For instance, the trajectory $\mathtt{turn}_{\mathbf{s},R,\alpha,\omega,v_z}$, which has the type $\mathbb{R} \to \mathbb{R}^3$, can be trivially

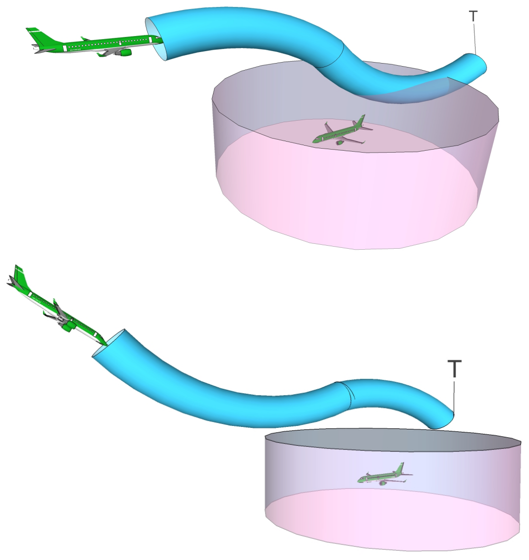

Figure 6: Conflict and Conflict-Free Scenarios

extended to a trajectory of type $\mathbb{R}^4 \to \mathbb{R}^3$, the same type as $\texttt{windturn}_{\mathbf{s},R,\alpha,\omega,v_z,E_w,E_a}$, by defining $\texttt{turn}'_{\mathbf{s},R,\alpha,\omega,v_z,E_w,E_a}(t,x,y,z) \equiv \texttt{turn}_{\mathbf{s},R,\alpha,\omega,v_z}(t)$.

3.2 Conflicts and Conflict Detection Algorithms

While loss of separation is formalized as a predicate on two aircraft positions $\mathbf{s}_o$ and $\mathbf{s}_i$, a conflict between two aircraft is formalized as a predicate on the ownship and intruder trajectories, P_o and P_i in $\mathbb{R}^m \to \mathbb{R}^3$, respectively, and a box $[\boldsymbol{a},\boldsymbol{b}] \in \mathbb{R}^m$ that represents a range of interest in $\mathbb{R}^m$, e.g., time should be in $[0,T]$. The trajectories P_o and P_i are in *conflict* on the box $[\boldsymbol{a},\boldsymbol{b}]$ if there exists a point $\boldsymbol{x} \in [\boldsymbol{a},\boldsymbol{b}]$ such that the positions $P_o(\boldsymbol{x})$ and $P_i(\boldsymbol{x})$ are in loss of separation:

$$\textit{conflict?}(D,H,\boldsymbol{a},\boldsymbol{b},P_o,P_i) \equiv \exists \boldsymbol{x} \in [\boldsymbol{a},\boldsymbol{b}] : \textit{los?}(D,H,P_o(\boldsymbol{x}),P_i(\boldsymbol{x})). \tag{5}$$

Figure 6 illustrates conflict and conflict-free scenarios in a relative coordinate system, where the intruder is at the origin of the system and the ownship is moving relative to the intruder. In each scenario, the "tube" in front of the ownship represents the relative trajectory of the aircraft for a given box $[\boldsymbol{a},\boldsymbol{b}] \in \mathbb{R}^m$, where the first component in the box the time interval $[0,T]$. When $\mathbb{R}^m = \mathbb{R}$, i.e., $m = 1$, the relative trajectory is a just a line.

An algorithm used by an aircraft to detect conflicts with another aircraft is called a *conflict detection algorithm*. In this paper, a conflict detection algorithm is a function `cd` that takes as inputs D, H, $\boldsymbol{a}$, $\boldsymbol{b}$, P_o, and P_i and returns an element of type `CDOutcome`, with possible values `LossAt`$(\boldsymbol{c})$, where $\boldsymbol{c} \in [\boldsymbol{a},\boldsymbol{b}]$, `NoConflict`, and `Unknown`.

Formally, a conflict detection algorithm `cd` is *sound* if for all positive real numbers D, H, boxes $[\boldsymbol{a},\boldsymbol{b}]$, and trajectories P_o, P_i, $\texttt{cd}(D,H,\boldsymbol{a},\boldsymbol{b},P_o,P_i) = \texttt{NoConflict}$ implies that $\neg$*conflict?*$(D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i)$ hold; it is *complete* if $\texttt{cd}(\boldsymbol{a}, \boldsymbol{b}, P_o, P_i) = \texttt{LossAt}(\boldsymbol{c})$ for some $\boldsymbol{c} \in [\boldsymbol{a}, \boldsymbol{b}]$ implies that *conflict?*$(D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i)$ holds. An algorithm that is *sound* and *complete* is said to be *correct*. It is clear from the definition that correctness is a key safety property for a conflict detection algorithm.

3.3 Conflict Detection and Global Optimization

Conflict between two trajectories can be written as an optimization problem and hence global optimization methods can be used for conflict detection. This problem transformation is accomplished through the cylindrical norm, which is described in detail in [12]. The *cylindrical norm* of a vector $\mathbf{w}$ is the quantity

$$\|\mathbf{w}\|_{D,H} \equiv \max\left(\frac{|w_z|}{H}, \frac{\|\mathbf{w}_{(x,y)}\|}{D}\right). \tag{6}$$

With this length, $\mathbb{R}^3$ is a metric space in the sense of real analysis [22], and it therefore satisfies the triangle inequality. The following lemma and theorems follow directly from definitions.

Lemma 3.1 *The ownship and the intruder, which have position vectors $\mathbf{s}_o$ and $\mathbf{s}_i$, are in loss of separation, i.e., los?$(D, H, \mathbf{s}_o, \mathbf{s}_i)$ holds, if and only if $\|\mathbf{s}_o - \mathbf{s}_i\|_{D,H} < 1$.*

Instead of the cylindrical norm, which involves the square root and absolute value functions, the square of the cylindrical norm is considered. The function $\|\mathbf{w}\|^2_{D,H}$, defined as follows

$$\|\mathbf{w}\|^2_{D,H} \equiv \max\left(\frac{w_z^2}{H^2}, \frac{w_x^2 + w_y^2}{D^2}\right),$$

is a maximum of two polynomials, the first in w_z, and the second in w_x and w_y. The following theorem shows that conflict between two trajectories can be expressed using this function.

Theorem 3.1 *For all boxes $[\boldsymbol{a}, \boldsymbol{b}]$, positive constant values D and H, and trajectories P_o and P_i, conflict?$(D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i)$ holds if and only if there exists $\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]$ such that $\texttt{sqdist}_{D,H,P_o,P_i}(\boldsymbol{x}) < 1$, where*

$$\texttt{sqdist}_{D,H,P_o,P_i}(\boldsymbol{x}) \equiv \max\left(\frac{w_z^2}{H^2}, \frac{w_x^2 + w_y^2}{D^2}\right), \text{ with } \mathbf{w} = P_o(\boldsymbol{x}) - P_i(\boldsymbol{x}). \tag{7}$$

When both P_o and P_i are polynomial trajectories, as in the cases of Example 1 and Example 2, the parametric function $\texttt{sqdist}$ is defined as the maximum between two polynomials. If the trajectories are not polynomials, such as the trajectories in Example 3 and Example 4, since these functions are continuous on $[\boldsymbol{a}, \boldsymbol{b}]$ they can be uniformly approximated by polynomials to any given precision. This result is known as the Weierstrass Approximation Theorem [9].

The predicate *approx?* specifies whether a given trajectory P is approximated by a trajectory P' on the box $[\boldsymbol{a}, \boldsymbol{b}]$ by a horizontal precision $\varepsilon_D \geq 0$ and a vertical precision $\varepsilon_H \geq 0$.

$$\begin{aligned} \textit{approx?}(\boldsymbol{a}, \boldsymbol{b}, \varepsilon_D, \varepsilon_H, P, P') \equiv \forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}] \colon & |P(\boldsymbol{x})_z - P'(\boldsymbol{x})_z| < \varepsilon_H \text{ and} \\ & \|P(\boldsymbol{x})_{(x,y)} - P'(\boldsymbol{x})_{(x,y)}\| < \varepsilon_D. \end{aligned}$$

If it is known that the trajectories P'_o and P'_i, which are approximations of P_o and P_i, respectively, are in conflict, then it is possible to determine if P_o and P_i are in conflict as well. This determination can be done by considering the effect that the differences between these trajectories and their approximations have on the output of the function `sqdist`. This effect is captured in the following proposition, which can be proved by basic algebraic manipulations.

Proposition 1 *Let $[\boldsymbol{a},\boldsymbol{b}]$, P_o, P'_o, P_i, P'_i, $\varepsilon_D = \varepsilon_{oD} + \varepsilon_{iD}$, and $\varepsilon_H = \varepsilon_{oH} + \varepsilon_{iH}$, be such that approx?$(\boldsymbol{a},\boldsymbol{b},\varepsilon_{oD},\varepsilon_{oH},P_o,P'_o)$ and approx?$(\boldsymbol{a},\boldsymbol{b},\varepsilon_{iD},\varepsilon_{iH},P_i,P'_i)$ hold. Then, for all $\boldsymbol{x} \in [\boldsymbol{a},\boldsymbol{b}]$,*

1. *$\texttt{sqdist}_{D,H,P'_o,P'_i}(\boldsymbol{x}) < 1 - \delta_-(D,H,\varepsilon_D,\varepsilon_H)$ implies $\texttt{sqdist}_{D,H,P_o,P_i}(\boldsymbol{x}) < 1$,*
2. *$\texttt{sqdist}_{D,H,P'_o,P'_i}(\boldsymbol{x}) \geq 1 + \delta_+(D,H,\varepsilon_D,\varepsilon_H)$ implies $\texttt{sqdist}_{D,H,P_o,P_i}(\boldsymbol{x}) \geq 1$,*

where

$$\begin{aligned}\delta_-(D,H,\varepsilon_D,\varepsilon_H) &\equiv \max(2\frac{\varepsilon_D}{D} - \frac{\varepsilon_D^2}{D^2}, 2\frac{\varepsilon_H}{H} - \frac{\varepsilon_H^2}{H^2}),\\ \delta_+(D,H,\varepsilon_D,\varepsilon_H) &\equiv \max(2\frac{\varepsilon_D}{D} + \frac{\varepsilon_D^2}{D^2}, 2\frac{\varepsilon_H}{H} + \frac{\varepsilon_H^2}{H^2}).\end{aligned} \tag{8}$$

The next result follows directly from Theorem 3.1 and Proposition 1. It shows that if P_o and P_i are trajectories that are approximated by P'_o and P'_i, then range information for the function $\texttt{sqdist}_{D,H,P'_o,P'_i}$ can be used to detect conflict between P_o and P_i. This result will be directly used later to detect conflicts between P_o and P_i, when P'_o and P'_i are polynomial trajectories, by approximating the range of $\texttt{sqdist}_{D,H,P'_o,P'_i}$. Since any trajectory can be uniformly approximated by polynomial trajectories, conflict detection for arbitrary trajectories can be reduced to computing range information for the function $\texttt{sqdist}_{D,H,P'_o,P'_i}$ for some polynomial trajectories P'_o and P'_i.

Proposition 2 *If approx?$(\boldsymbol{a},\boldsymbol{b},\varepsilon_{oD},\varepsilon_{oH},P_o,P'_o)$ and approx?$(\boldsymbol{a},\boldsymbol{b},\varepsilon_{iD},\varepsilon_{iH},P_i,P'_i)$, then*

1. *If $\boldsymbol{c}$ is a point in the box $[\boldsymbol{a},\boldsymbol{b}]$ and $\texttt{sqdist}_{D,H,P'_o,P'_i}(\boldsymbol{c}) < 1 - \delta_-(D,H,\varepsilon_D,\varepsilon_H)$, then conflict?$(D,H,\boldsymbol{a},\boldsymbol{b},P_o,P_i)$.*
2. *conflict?$(D,H,\boldsymbol{a},\boldsymbol{b},P_o,P_i)$ implies $\texttt{sqdist}_{D,H,P'_o,P'_i}(\boldsymbol{x}) \leq 1 + \delta_+(D,H,\varepsilon_D,\varepsilon_H)$ for some $\boldsymbol{x} \in [\boldsymbol{a},\boldsymbol{b}]$,*

Numerical approximation methods based on Bernstein polynomials are well-known by the global optimization community [3]. The next section describes a formalization of Bernstein polynomials in PVS, from an algorithmic point of view. A formally verified algorithm for finding range information for the maximum and minimum values of a function defined as the maximum of two polynomials is presented in Section 5. That algorithm is used in verified algorithms for conflict detection of arbitrary trajectories, proposed in Section 6, the correctness of which follows from Proposition 2.

4 Formalization of Bernstein Polynomials

Bernstein polynomials are mathematical objects used to approximate continuous functions. This section presents an algorithmic formalization in PVS of multivariate Bernstein polynomials and their main properties. For technical details about this formalization, the reader is referred to [11].

Let $\boldsymbol{i}$ be an m-tuple of natural numbers and $\boldsymbol{x}$ be an m-tuple of variables over $\mathbb{R}$. The product $\boldsymbol{x}^{\boldsymbol{i}} = x_0^{i_0} \cdot \ldots \cdot x_{m-1}^{i_{m-1}}$ is called an *m-variable monomial.* The m-tuple $\boldsymbol{i}$ is called the *index* of the monomial $\boldsymbol{x}^{\boldsymbol{i}}$. An *m-variable polynomial* of degree $\boldsymbol{n}$ is a finite sum of the form

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c_{\boldsymbol{i}}\, \boldsymbol{x}^{\boldsymbol{i}},$$

where the elements $c_{\boldsymbol{i}} \in \mathbb{R}$ are called the *coefficients* of p. In PVS, an m-polynomial p is represented by a function from $\mathbb{R}^m$ into $\mathbb{R}$.

Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c_{\boldsymbol{i}}\, \boldsymbol{x}^{\boldsymbol{i}}$ be an m-variable polynomial. For any bounded box $[\boldsymbol{a}, \boldsymbol{b}]$, the m-polynomial $p_{[\boldsymbol{a},\boldsymbol{b}]}$ can be defined as follows

$$p_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{x}) \equiv \sum_{\boldsymbol{k} \leq \boldsymbol{n}} \sum_{\boldsymbol{k} \leq \boldsymbol{i} \leq \boldsymbol{n}} (c_{\boldsymbol{i}} \prod_{j=0}^{m-1} \binom{i_j}{k_j} (b_j - a_j)^{k_j} a_j^{i_j - k_j})\, \boldsymbol{x}^{\boldsymbol{k}}. \tag{9}$$

The PVS formalization provides a function `translate` that takes as inputs the arrays $\boldsymbol{a}$ and $\boldsymbol{b}$, and an m-polynomial p, and returns the polynomial $p_{[\boldsymbol{a},\boldsymbol{b}]}$ defined by Formula (9). The function `translate` satisfies

$$p(\sigma_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{x})) = \texttt{translate}(\boldsymbol{a}, \boldsymbol{b}, p)(\boldsymbol{x}),$$
$$p(\boldsymbol{y}) = \texttt{translate}(\boldsymbol{a}, \boldsymbol{b}, p)(\sigma^{-1}_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{y})),$$

where $\sigma_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{x})_j \equiv a_j + x_j(b_j - a_j)$. It follows directly from this that the maximum (resp. minimum) value attained by p on the box $[\boldsymbol{a}, \boldsymbol{b}]$ is the maximum (resp. minimum) value attained by the polynomial $\texttt{translate}(\boldsymbol{a}, \boldsymbol{b}, p)$ on the unit box.

4.1 Bernstein Polynomials

An m-variable Bernstein polynomial is an m-variable polynomial of the form

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x}), \tag{10}$$

where $\hat{b}_{\boldsymbol{i}} \in \mathbb{R}$ and

$$B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x}) \equiv \prod_{j=0}^{m-1} \binom{n_j}{i_j} x_j^{i_j} (1 - x_j)^{n_j - i_j}. \tag{11}$$

The coefficients $\hat{b}_{\boldsymbol{i}}$ are called the *Bernstein coefficients* of p. Any polynomial can be written as a polynomial in Bernstein form by a simple transformation. Thus, the m-variable polynomials $B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ in Formula (11) form a basis for the vector space of m-variable polynomials of degree at least $\boldsymbol{n}$.

In PVS, m-variable polynomials in Bernstein form are represented using the data structure proposed by Smith in [23]. Since the method presented in this paper does not depend on a particular representation of polynomials, it suffices to say that the PVS formalization provides a function `tomultibern` that takes an m-variable polynomial p as input and returns an element of a particular data structure that represents the Bernstein form of p. Furthermore, a function `eval` that takes as inputs a representation of an m-variable polynomial in Bernstein form and an m-tuple $\boldsymbol{x}$, and returns a real number is defined such that for all $\boldsymbol{x} \in \mathbb{R}^m$,

$$\texttt{eval}(\texttt{tomultibern}(p), \boldsymbol{x}) = p(\boldsymbol{x}). \tag{12}$$

A key result that makes Bernstein polynomials useful for proving polynomial inequalities is that the Bernstein coefficients of a polynomial provide lower and upper bounds for the values of the polynomial over the unit box. Another useful property of Bernstein polynomials is that the values of the function at the endpoints of the unit box are Bernstein coefficients of the polynomial. These properties are summarized in the following inequalities,

$$\begin{aligned}\min_{\boldsymbol{i}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{i}} \le \min_{\boldsymbol{x}\in[\mathbf{0},\mathbf{1}]} p(\boldsymbol{x}) \le \min_{\boldsymbol{i}\in\mathcal{C}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}},\\ \max_{\boldsymbol{i}\in\mathcal{C}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}} \le \max_{\boldsymbol{x}\in[\mathbf{0},\mathbf{1}]} p(\boldsymbol{x}) \le \max_{\boldsymbol{i}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{i}},\end{aligned} \tag{13}$$

where $\boldsymbol{n}$ is an m-tuple of natural numbers and $\mathcal{C}_{\boldsymbol{n}}$ denotes the set of *endpoint indices* of $\boldsymbol{n}$, i.e., defined as follows.

$$\mathcal{C}_{\boldsymbol{n}} \equiv \{\boldsymbol{i} \le \boldsymbol{n} \mid \forall\ 0 \le j < m\ :\ \ i_j = 0 \text{ or } i_j = n_j\}. \tag{14}$$

The record type

$$\begin{aligned}\texttt{Outminmax} \equiv \texttt{lbmin}:\texttt{real} \times \texttt{lbmax}:\texttt{real} \times \texttt{lbvar}:\mathbb{R}^{m+} \times\\ \texttt{ubmax}:\texttt{real} \times \texttt{ubmin}:\texttt{real} \times \texttt{ubvar}:\mathbb{R}^{m+},\end{aligned}$$

defined in PVS, stores information about the range of a function $f\colon \mathbb{R}^m \to \mathbb{R}$ over a given box. The type $\mathbb{R}^{m+}$ is the type of points in $\mathbb{R}^m$ extended with special value that represents an empty tuple. The intended semantics of the type `Outminmax` is given by the predicate *sound?*. This predicate is defined on a function $f\colon \mathbb{R}^m \to \mathbb{R}$, an element `omm` of type `Outminmax`, and a box $[\boldsymbol{a}, \boldsymbol{b}]$ such that *sound?*$(f, \texttt{omm}, \boldsymbol{a}, \boldsymbol{b})$ holds if and only if

- `omm.lbmin` and `omm.lbmax` are, respectively, minimum and maximum possible values for the lower bound of f on the box $[\boldsymbol{a}, \boldsymbol{b}]$.
- `omm.lbvar` is either empty or is a point in the box $[\boldsymbol{a}, \boldsymbol{b}]$ where f attains the value `omm.lbmax`.
- `omm.ubmin` and `omm.ubmax` are, respectively, minimum and maximum possible values for the upper bound of f on the box $[\boldsymbol{a}, \boldsymbol{b}]$.
- `omm.ubvar` is either empty or is a point in the box $[\boldsymbol{a}, \boldsymbol{b}]$ where f attains the value `omm.ubmin`.

Thus, if *sound?*$(f, \texttt{omm}, \boldsymbol{a}, \boldsymbol{b})$ holds, then `omm` provides bounds on the minimum and maximum values of f on $[\boldsymbol{a}, \boldsymbol{b}]$.

The PVS formalization provides a function `berncoeffsminmax` that takes as input a representation of an m-variable polynomial in Bernstein form $p(\boldsymbol{x}) = \sum_{\boldsymbol{i}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ and returns an element `omm` of type `Outminmax` such that

- `omm.lbmin` $= \min_{\boldsymbol{i}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{i}}$, `omm.lbmax` $= \min_{\boldsymbol{i}\in\mathcal{C}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}}$, $p(\texttt{omm.lbvar}) = \texttt{lbmax}$, and
- `omm.ubmin` $= \max_{\boldsymbol{i}\in\mathcal{C}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}}$, `omm.ubmax` $= \max_{\boldsymbol{i}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{i}}$, $p(\texttt{omm.ubvar}) = \texttt{lbmin}$.

The next proposition follows directly from the definition of `berncoeffsminmax`.

Proposition 3 *For any m-variable polynomial p,*

$$\textit{sound?}(p, \textit{berncoeffsminmax}(\textit{tomultibern}(p)), \mathbf{0}, \mathbf{1})$$

holds.

Let p be an m-variable polynomial defined on a box $[\boldsymbol{a}, \boldsymbol{b}]$. When the function `berncoeffsminmax` is evaluated on `tomultibern`(`translate`$(\boldsymbol{a}, \boldsymbol{b}, p)$) and the unit box $[\mathbf{0}, \mathbf{1}]$, the fields `lbvar` and `ubvar` of the resulting element of type `Outminmax` are points in $[\mathbf{0}, \mathbf{1}]$ where the m-variable polynomial p attains the values `lbmax` and `ubmin`, respectively.

Proposition 4 *For any m-variable polynomial p and box $[\boldsymbol{a}, \boldsymbol{b}]$, if*

$$omm = \mathtt{berncoeffsminmax}(\mathtt{tomultibern}(\mathtt{translate}(\boldsymbol{a}, \boldsymbol{b}, p))),$$

then sound?$(p$, denorm_omm$(omm, \boldsymbol{a}, \boldsymbol{b}), \boldsymbol{a}, \boldsymbol{b})$ holds, where

$$\begin{aligned}\mathit{denorm\_omm}(omm, \boldsymbol{a}, \boldsymbol{b}) \equiv omm\ \mathtt{with}\,[\ & \mathtt{lbvar} \leftarrow \sigma_{[\boldsymbol{a},\boldsymbol{b}]}(omm.\mathtt{lbvar}),\\ & \mathtt{ubvar} \leftarrow \sigma_{[\boldsymbol{a},\boldsymbol{b}]}(omm.\mathtt{ubvar})\,].\end{aligned}$$

According to Proposition 4, the range information for a polynomial p over an arbitrary box $[\boldsymbol{a}, \boldsymbol{b}]$ can be computed using the functions `translate`, `tomultibern`, `berncoeffsminmax`, and `denorm_omm`.

4.2 Domain Subdivision

The lower and upper bounds of the minimum and maximum values of a polynomial on a bounded box given by Formula (13) are not always exact. There is, however, a method that can be used to significantly improve the accuracy of the estimates for the minimum and maximum values of a multivariate polynomial p in a bounded box $[\boldsymbol{a}, \boldsymbol{b}]$. The basic idea is to subdivide $[\boldsymbol{a}, \boldsymbol{b}]$ into two boxes by picking a variable x_j, where $j < m$, and consider the case where $a_j \leq x_j \leq \frac{a_j+b_j}{2}$ separately from the case where $\frac{a_j+b_j}{2} \leq x_j \leq b_j$. This method can be used recursively to compute arbitrarily precise bounds of the minimum and maximum values of the polynomial on $[\boldsymbol{a}, \boldsymbol{b}]$.

The de Casteljau algorithm [4] is commonly used to compute the Bernstein forms of $p(\boldsymbol{x}\ \mathtt{with}\,[\,j \leftarrow \frac{x_j}{2}\,])$ and $p(\boldsymbol{x}\ \mathtt{with}\,[\,j \leftarrow \frac{x_j+1}{2}\,])$. In PVS, functions `subdivl` and `subdivr` are defined that take as inputs a representation of an m-variable polynomial in Bernstein form p and an index $j < m$, and return the representations of the m-variable polynomials in Bernstein form for the polynomials $p(\boldsymbol{x}\ \mathtt{with}\,[\,j \leftarrow \frac{x_j}{2}\,])$ and $p(\boldsymbol{x}\ \mathtt{with}\,[\,j \leftarrow \frac{x_j+1}{2}\,])$, respectively. The functions `subdivl` and `subdivr` satisfy the following proposition.

Proposition 5 *For any m-variable polynomial p and for all $\boldsymbol{x} \in \mathbb{R}^m$,*

$$\begin{aligned}\mathtt{eval}(\mathtt{subdivl}(\mathtt{tomultibern}(p), j)(\boldsymbol{x}) &= p(\boldsymbol{x}\ \mathtt{with}\,[\,j \leftarrow \frac{x_j}{2}\,]),\\ \mathtt{eval}(\mathtt{subdivr}(\mathtt{tomultibern}(p), j)(\boldsymbol{x}) &= p(\boldsymbol{x}\ \mathtt{with}\,[\,j \leftarrow \frac{x_j+1}{2}\,]).\end{aligned}$$

The functions `subdivl` and `subdivr` can be used to improve the accuracy of the estimates for the minimum and maximum values of a polynomial in Bernstein form p on the unit box. This result is captured in the following proposition.

Proposition 6 *Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ be an m-variable polynomial in Bernstein form, K be a real number, and $\Re$ be a real order in $\{\leq, <, \geq, >\}$. If $\hat{b}^L_{\boldsymbol{i}}\ \Re\ K$ and $\hat{b}^R_{\boldsymbol{i}}\ \Re\ K$, for all $\boldsymbol{i} \leq \boldsymbol{n}$, then $p(\boldsymbol{x})\ \Re\ K$, for all $\boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]$.*

The function `berncoeffsminmax`, when applied to $\texttt{subdivl}(\texttt{tomultibern}(p), j)$ and $\texttt{subdivr}(\texttt{tomultibern}(p), j)$, returns elements of type `Outminmax` that represent range information for the polynomial p on the "left" and "right" (in the j-th variable) halves of the unit box, respectively. The function `combine` is used to aggregate the information from these elements of `Outminmax` into one element of `Outminmax` that represents range information for the polynomial p on the entire unit box $[\mathbf{0}, \mathbf{1}]$. Given elements $\texttt{omm}_1$ and $\texttt{omm}_2$ of type `Outminmax`, $\texttt{combine}(\texttt{omm}_1, \texttt{omm}_2)$ returns an element `omm` that satisfies

- $\texttt{omm.lbmin} = \min(\texttt{omm}_1\texttt{.lbmin}, \texttt{omm}_2\texttt{.lbmin})$,
- $\texttt{omm.lbmax} = \min(\texttt{omm}_1\texttt{.lbmax}, \texttt{omm}_2\texttt{.lbmax})$,
- `omm.lbvar` is $\texttt{omm}_1\texttt{.lbvar}$ if $\texttt{omm.lbmax} = \texttt{omm}_1\texttt{.lbmax}$, otherwise $\texttt{omm.lbvar} = \texttt{omm}_2\texttt{.lbvar}$,
- $\texttt{omm.ubmin} = \max(\texttt{omm}_1\texttt{.ubmin}, \texttt{omm}_2\texttt{.ubmin})$,
- $\texttt{omm.ubmax} = \max(\texttt{omm}_1\texttt{.ubmax}, \texttt{omm}_2\texttt{.ubmax})$, and
- `omm.ubvar` is $\texttt{omm}_1\texttt{.ubvar}$ if $\texttt{omm.ubmax} = \texttt{omm}_1\texttt{.ubmax}$, otherwise $\texttt{omm.ubvar} = \texttt{omm}_2\texttt{.ubvar}$.

Let `p` be a polynomial in Bernstein form that represents the polynomial p, i.e., $\texttt{p} = \texttt{tomultibern}(p)$. It is not true that combining the outputs of $\texttt{subdivl}(\texttt{p}, j)$ and $\texttt{subdivr}(\texttt{p}, j)$ produces an element of `Outminmax` that satisfies the predicate *sound?*. The fields `lbvar` and `ubvar` in $\texttt{omm}_l = \texttt{berncoeffsminmax}(\texttt{subdivl}(\texttt{p}, j))$ and $\texttt{omm}_r = \texttt{berncoeffsminmax}(\texttt{subdivr}(\texttt{p}, j))$ are points in the unit box where the polynomials represented by $\texttt{subdivl}(\texttt{p}, j)$ and $\texttt{subdivr}(\texttt{p}, j)$ attain the values `lbmax` and `ubmin`, respectively. However, each of these polynomials represents p on only half of the unit box $[\mathbf{0}, \mathbf{1}]$. Thus, the fields $\texttt{omm}_l\texttt{.lbvar}$, $\texttt{omm}_l\texttt{.ubvar}$, $\texttt{omm}_r\texttt{.lbvar}$, and $\texttt{omm}_r\texttt{.ubvar}$ have to be translated from $[\mathbf{0}, \mathbf{1}]$ back to their respective half intervals for the predicate *sound?* to hold. This translation is accomplished through the functions `update` and `updateOutminmax`. The function `update` takes as inputs a tuple $\boldsymbol{c} \in \mathbb{R}^m$, a function $u \colon \mathbb{R} \to \mathbb{R}$, and a variable index $j < m$. It returns a tuple that is equal to $\boldsymbol{c}$ in all components but the j-th component where it has the value $u(c_j)$, i.e.,

$$\texttt{update}(\boldsymbol{c}, u, j) \equiv \boldsymbol{c} \texttt{ with} \, [\, j \leftarrow u(c_j) \,].$$

The function `updateOutminmax` takes an element `omm` of type `Outminmax`, a function $u \colon \mathbb{R} \to \mathbb{R}$, and a variable index $j < m$ as inputs. It updates the j-th components of the m-tuples `omm.lbvar` and `omm.ubvar` using the function u:

$$\begin{aligned}\texttt{updateOutminmax}(\texttt{omm}, u, j) \equiv \texttt{omm with} \, [\, & \texttt{lbvar} \leftarrow \texttt{update}(\texttt{omm.lbvar}, u, j), \\ & \texttt{ubvar} \leftarrow \texttt{update}(\texttt{omm.ubvar}, u, j) \,].\end{aligned}$$

Proposition 7 *For any m-variable polynomial p, variable index $j < m$, and elements $\texttt{omm}_l$, $\texttt{omm}_r$ of type* `Outminmax` *such that*

$$\begin{aligned}\texttt{omm}_l &= \texttt{updateOutminmax}(\texttt{subdivl}(\texttt{tomultibern}(p), j), \lambda x \colon \frac{x}{2}, j), \\ \texttt{omm}_r &= \texttt{updateOutminmax}(\texttt{subdivr}(\texttt{tomultibern}(p), j), \lambda x \colon \frac{x+1}{2}, j),\end{aligned}$$

sound?$(p, \texttt{combine}(\texttt{omm}_l, \texttt{omm}_r), \mathbf{0}, \mathbf{1})$ *holds.*

5 The Maximum of Two Polynomials

Theorem 3.1 in Section 3.3 shows that the problem of conflict detection for polynomial trajectories is equivalent to finding the global minimum of a function that is defined as the maximum of two polynomials.

If f and g are two functions from $\mathbb{R}^m$ into $\mathbb{R}$, then the function $\max(f, g)$ from $\mathbb{R}^m$ into $\mathbb{R}$ is defined by

$$\max(f, g)(\boldsymbol{x}) \equiv \max(f(\boldsymbol{x}), g(\boldsymbol{x})).$$

If the functions f and g are polynomials, then the function `berncoeffsminmax` discussed in Section 4.1 can be used to compute an element of `Outminmax` that satisfies the predicate *sound?* on the unit box for the function $\max(f, g)$. This is accomplished through the function

$$\begin{aligned}\texttt{max\_bc\_minmax}(\texttt{p}, \texttt{q}) \equiv \texttt{combine\_max}(&\texttt{berncoeffsminmax}(\texttt{p}),\\ &\texttt{berncoeffsminmax}(\texttt{q})),\end{aligned}$$

where $\texttt{p} = \texttt{tomultibern}(f)$ and $\texttt{q} = \texttt{tomultibern}(g)$. In the definition of this function, `berncoeffsminmax` is used to compute two elements of `Outminmax` that contain range information for f and g on $[\mathbf{0}, \mathbf{1}]$. The function `combine_max` then computes the worst case scenario for the range of the function $\max(f, g)$ on $[\mathbf{0}, \mathbf{1}]$, and it is defined as follows.

$$\texttt{combine\_max}(\texttt{omm}_1, \texttt{omm}_2) \equiv \begin{cases} \texttt{omm}_1 & \text{if } \texttt{omm}_1.\texttt{lbmin} \geq \texttt{omm}_2.\texttt{ubmax}, \\ \texttt{omm}_2 & \text{if } \texttt{omm}_2.\texttt{lbmin} > \texttt{omm}_1.\texttt{ubmax}, \\ \texttt{omm} & \text{otherwise,} \end{cases}$$

where $\texttt{omm.lbmin} = \min(\texttt{omm}_1.\texttt{lbmin}, \texttt{omm}_2.\texttt{lbmin})$, $\texttt{omm.lbmax} = 0$, $\texttt{omm.ubmin} = 0$, $\texttt{omm.ubmax} = \max(\texttt{omm}_1.\texttt{ubmax}, \texttt{omm}_2, \texttt{ubmax})$, and both `omm.lbvar` and `omm.ubvar` are set to empty.

Proposition 8 *For any m-variable polynomials p and q,*

$$\textit{sound?}(\max(p, q), \textit{max\_bc\_minmax}(\texttt{p}, \texttt{q}), \mathbf{0}, \mathbf{1})$$

holds, where $\texttt{p} = \texttt{tomultibern}(p)$ and $\texttt{q} = \texttt{tomultibern}(q)$.

Proposition 8 states that the function `max_bc_minmax` computes bounds on the minimum and maximum values of the function $\max(p, q)$ in the unit box. Although these bounds are correct, they may not be precise enough to determine whether a relation such as $\max(p, q) \geq K$ is satisfied for a given real number K. In this case, Proposition 6 can be recursively applied to subdivide the unit box into smaller intervals and compute estimates that are precise to any required approximation.

5.1 A Procedure For Checking $\max(p, q) \geq K$ in $[\mathbf{0}, \mathbf{1}]$

The following procedure can be used to determine whether the maximum $\max(p, q)$ of two m-variable polynomials p and q always takes a value of at least K on the unit box $[\mathbf{0}, \mathbf{1}]$.

1. Compute $\texttt{omm} = \texttt{max\_bc\_minmax}(\texttt{tomultibern}(p), \texttt{tomultibern}(q))$.
2. If $\texttt{omm.lbmin} \geq K$, then the inequality $\max(p, q)(\boldsymbol{x}) \geq K$ holds for all $\boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]$. The procedure exits with success.

3. If `omm.lbmax` $< K$, then the inequality $\max(p,q)(\boldsymbol{c}) < K$ holds for $\boldsymbol{c} =$ `omm.lbvar` $\in [\mathbf{0},\mathbf{1}]$. The procedure exits with failure, with $\boldsymbol{c}$ as counterexample.
4. Otherwise, choose any $0 \leq j < m$ and recursively apply this procedure to determine whether
   (a) $\max(p,q)(\boldsymbol{x}\ \texttt{with}\,[\,j \leftarrow \frac{x_j}{2}\,]) \geq K$ and
   (b) $\max(p,q)(\boldsymbol{x}\ \texttt{with}\,[\,j \leftarrow \frac{x_j+1}{2}\,]) \geq K$,

   for all $\boldsymbol{x} \in [\mathbf{0},\mathbf{1}]$.
   - If both Step 4a and Step 4b exit with success, then, by Proposition 6, the inequality $\max(p,q)(\boldsymbol{x}) \geq K$ holds for all $\boldsymbol{x} \in [\mathbf{0},\mathbf{1}]$. The procedure exits with success.
   - If Step 4a exits with failure with counterexample $\boldsymbol{c}$, then the inequality $\max(p,q)(\boldsymbol{c}_l) < K$ holds, where $\boldsymbol{c}_l = \boldsymbol{c}\ \texttt{with}\,[\,j \leftarrow \frac{x_j}{2}\,] \in [\mathbf{0},\mathbf{1}]$. The process exits with failure, with $\boldsymbol{c}_l$ as counterexample.
   - If Step 4b exits with failure with counterexample $\boldsymbol{c}$, then the inequality $\max(p,q)(\boldsymbol{c}_r) < K$ holds, where $\boldsymbol{c}_r = \boldsymbol{c}\ \texttt{with}\,[\,j \leftarrow \frac{x_j+1}{2}\,] \in [\mathbf{0},\mathbf{1}]$. The process exits with failure, with $\boldsymbol{c}_r$ as counterexample.

It should be noted that the procedure above does not necessarily terminate. However, at each recursive step, the interval [`omm.lbmin`, `omm.lbmax`], which contains the minimum value of $\max(p,q)$ over the unit box $[\mathbf{0},\mathbf{1}]$, gets smaller.

Given m-variable polynomials p and q and a real number K, the procedure above can also be used to determine whether the inequality $\max(p,q)(\boldsymbol{x}) \geq K$ holds for all $\boldsymbol{x} \in [\boldsymbol{a},\boldsymbol{b}]$. In this case, the procedure is used with the m-variable polynomials $p_{[\boldsymbol{a},\boldsymbol{b}]} = \texttt{translate}(\boldsymbol{a},\boldsymbol{b},p)$ and $q_{[\boldsymbol{a},\boldsymbol{b}]} = \texttt{translate}(\boldsymbol{a},\boldsymbol{b},q)$. If the procedure exits with success, then the inequality $\max(p,q)(\boldsymbol{x}) \geq K$ holds for all $\boldsymbol{x} \in [\boldsymbol{a},\boldsymbol{b}]$. If the procedure exits with failure with counterexample $\boldsymbol{c}$, the inequality $\max(p,q)(\boldsymbol{c}') < K$ holds, where $\boldsymbol{c}' = \sigma_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{c})$.

The complexity of the procedure is exponential in the number of variables. However, some heuristics can be used to greatly speed this procedure [23, 20, 16]. For instance, Step 4 involves applying the procedure recursively on the left and right hand sides of the unit box. However, if the procedure is run for the left (resp. right) side of the unit box first, in some cases, it may not be necessary to run it for the right (resp. left) hand side at all. An example of this is when the recursive call on the right (resp. left) hand side exits with failure with a counter example $\boldsymbol{c}$. In this case, $\boldsymbol{c}$ is also a counterexample for the larger box, so running the procedure on the left (resp. right) hand side of the box is not necessary. There are also heuristics for the selection of the variable j in Step 4 and of the order in which Step 4a and Step 4b are performed. These heuristics improve the efficiency of the procedure by pruning the execution tree generated by the recursive calls in Step 4.

5.2 The Algorithm `bernMinmax`

This section describes a formally verified algorithm, which is based on the procedure presented in Section 5.1, that computes range information for the minimum and maximum values of a function defined as the maximum of two polynomials.

The function `bernMinmax`, defined in Figure 7, has as inputs representations `p` and `q` of m-variable polynomials in Bernstein form p and q, respectively, a maximum

```
bernMinmax(p, q, N, i, varsel, localex, globalex, omm): Outminmax $\equiv$
  let bmm = max_bc_minmax(p, q) in
    if i = N or localex(bmm) or (i $>$ 0 and between(omm, bmm)) or
        globalex(bmm) then bmm
    else
      let (left$_p$, j$_p$) = varsel(p, i),
          (left$_q$, j$_q$) = varsel(q, i),
          (left, j) = if mod(i, 2) = 0 then (left$_p$, j$_p$)
                                  else (left$_q$, j$_q$) endif,
          (pl, pr) = (subdivl(p, j), subdivr(p, j)),
          (ql, qr) = (subdivl(q, j), subdivr(q, j)),
          (p$_1$, p$_2$) = if left then (pl, pr) else (pr, pl) endif,
          (q$_1$, q$_2$) = if left then (ql, qr) else (qr, ql) endif,
          $\sigma$ = if left then $\lambda x\colon x/2$ else $\lambda x\colon (x+1)/2$ endif,
          omm = if i $>$ 0 then combine(omm, bmm) else bmm endif,
          bmm$_1$ = bernMinmax(p$_1$, q$_1$, N, i + 1, varsel, omm) in
        if globalex(bmm$_1$) then
          combine(updateOutminmax(bmm$_1$, $\sigma$, j), bmm)
        else
          let omm = combine(omm, bmm$_1$),
              bmm$_2$ = bernMinmax(p$_2$, q$_2$, N, i + 1, varsel, omm),
              bmmleft = if left then bmm$_1$ else bmm$_2$ endif,
              bmmright = if left then bmm$_2$ else bmm$_1$ endif in
          combine(updateOutminmax(bmmleft, $\lambda x\colon x/2$, j),
                  updateOutminmax(bmmright, $\lambda x\colon (x+1)/2$, j))
        endif
    endif
```

Figure 7: The function bernMinmax

recursion depth $\mathtt{N} \in \mathbb{N}$, and the current recursion depth $\mathtt{i} \leq \mathtt{N}$. Additional inputs include a function `varsel` that selects the variable on which to subdivide at each iteration and which direction to explore first, predicates `localex` and `globalex` on the output type that cause the algorithm to exit locally and globally, respectively, and a parameter `omm` of the same type as the output value. These additional inputs allow for the use of heuristics based on a particular variable selection method, direction of recursion, and strategy for early termination. They are described in Section 5.4.

The function `bernMinmax` returns a record of type `Outminmax`. Assuming that $\mathtt{bmm} = \mathtt{max\_bc\_minmax}(\mathtt{p}, \mathtt{q})$ as in Line 1 of this function, if the condition in Line 3 is true, then the function returns `bmm`. In this case, `bmm` will satisfy either `localex` or `globalex`, except when the maximum depth has been reached, i.e., $\mathtt{i} = \mathtt{N}$, or when the execution tree is pruned by the condition $\mathtt{between}(\mathtt{omm}, \mathtt{bmm})$.

If the condition in Line 3 is false, then the function `varsel` is used to select a variable to subdivide and a direction (left or right) for each one of the polynomials. It is important to note that the function `varsel` is an input to `bernMinmax`, so it can handle any subdivision scheme. Next, the domain subdivision functions `subdivl` and `subdivr` presented in Section 4.2, are used to subdivide the unit box $[\mathbf{0}, \mathbf{1}]$ into smaller sub-boxes. At each subdivision, the function `max_bc_minmax` is recursively called to compute an element of `Outminmax` that stores information about the ranges of the polynomials on the given sub-box. Using this function, two elements $\mathtt{bmm}_1$ and $\mathtt{bmm}_2$ of `Outminmax` are produced; one representing range information for the box $[\mathbf{0}, \mathbf{1}\ \mathtt{with}\,[\,j \leftarrow \frac{1}{2}\,]]$ and the other representing range information for the box $[\mathbf{0}\ \mathtt{with}\,[\,j \leftarrow \frac{1}{2}\,], \mathbf{1}]$. The algorithm effectively stops when the predicate `globalex` is satisfied after the first of the two recursive calls of the function `bernMinmax`. This is acoomplished by the condition in Line 17. Since the points represented by `lbvar` and `ubvar` are computed in a unit box, they must be translated back to the half boxes from the full boxes in the function, by using the function `updateOutminmax`.

If the condition in Line 17 is false, the two elements of type `Outminmax` resulting from applying `updateOutminmax` to $\mathtt{bmm}_1$ and $\mathtt{bmm}_2$ are combined into a new element of type `Outminmax` that represents the range information of the function $\max(p, q)$ over the unit box.

The correctness property of the function `bernMinmax` states that it computes an element of type `Outminmax` that bounds the range of the function $\max(p, q)$ over the unit box. The following theorem has been proved in PVS by induction on the structure of the definition of `bernMinmax`. Proposition 8 is used to prove the base case. The inductive case is discharged by Proposition 7 in Section 4.2.

Theorem 5.1 *For all m-variable polynomials $p, q \colon \mathbb{R}^m \to \mathbb{R}$, $N \in \mathbb{N}$, $i \in \mathbb{N}$, with $i \leq N$, $\mathtt{varsel} \colon \mathbb{N} \to \mathtt{boolean} \times \mathbb{N}_{<m}$, $\mathtt{localex}, \mathtt{globalex} \colon \mathtt{Outminmax} \to \mathtt{boolean}$, and $\mathtt{omm} \in \mathtt{Outminmax}$, if $\mathtt{bmm} \in \mathtt{Outminmax}$ is given by*

$$\mathtt{bmm} = \mathtt{bernMinmax}(\mathtt{tomultibern}(p), \mathtt{tomultibern}(q), N, i, \mathtt{varsel}, \mathtt{omm}),$$

then

1. $\max(p, q)(\mathtt{bmm.lbvar}) = \mathtt{bmm.lbmax}$,
2. $\max(p, q)(\mathtt{bmm.ubvar}) = \mathtt{bmm.ubmin}$, *and*
3. $\mathtt{bmm.lbmin} \leq \max(p, q)(\boldsymbol{x}) \leq \mathtt{bmm.ubmax}$, *for all* $\boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]$.

It is noted that Theorem 5.1 holds for all possible values of the input parameters `varsel`, `localex`, `globalex`, and `omm`. These parameters are added for practical and efficiency reasons. They are explained in Section 5.4.

```
polyMinmax($p, q, \boldsymbol{a}, \boldsymbol{b}$, N, varsel, localex, globalex): Outminmax $\equiv$
    let  p = tomultibern(translate($p, \boldsymbol{a}, \boldsymbol{b}$)),
         q = tomultibern(translate($q, \boldsymbol{a}, \boldsymbol{b}$)),
       omm = bernMinmax(p, q, N, 0, varsel, localex, globalex, emptymm)
    in
      denorm_omm(omm, $\boldsymbol{a}, \boldsymbol{b}$)
```

Figure 8: The function `polyMinmax`

5.3 Function `polyMinmax`

The function `polyMinmax`, defined in Figure 8, computes range information for the maximum of two polynomials on an arbitrary box $[\boldsymbol{a}, \boldsymbol{b}]$. The algorithm works in four steps:

1. Convert the polynomials from the box $[\boldsymbol{a}, \boldsymbol{b}]$ to the unit box $[\mathbf{0}, \mathbf{1}]$ using the function `translate` defined by Formula (4) in Section 4.
2. Compute the Bernstein form of the translated polynomial using the function `tomultibern` from Section 4.1.
3. Apply `bernMinmax` to compute an element `bmm` of `Outminmax` that gives range information for the maximum of these two polynomials in Bernstein form on the unit box.
4. Translate `omm` from $[\mathbf{0}, \mathbf{1}]$ back to $[\boldsymbol{a}, \boldsymbol{b}]$ linearly using the function `denorm_omm` defined in Section 4.1.

The constant element `emptymm` of type `Outminmax` is defined such that all the numerical fields are 0 and the m-tuples are empty. The following correctness property of the function `polyMinmax` has been proved in PVS.

Theorem 5.2 *For all m-variable polynomials $p, q\colon \mathbb{R}^m \to \mathbb{R}$ in standard form, $N \in \mathbb{N}$, $varsel\colon \mathbb{N} \to boolean \times \mathbb{N}_{<m}$, and $localex, globalex\colon Outminmax \to boolean$, if $omm \in Outminmax$ is given by*

$$omm = polyMinmax(p, q, \boldsymbol{a}, \boldsymbol{b}, N, varsel, localex, globalex),$$

then

- $\max(p,q)(omm.lbvar) = omm.lbmax$, $\max(p,q)(omm.ubvar) = omm.ubmin$, *and*
- $omm.lbmin \le \max(p,q)(\boldsymbol{x}) \le omm.ubmax$, *for all $\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]$.*

According to Theorem 5.2, the range information computed by `polyMinmax` can be used to check universally quantified propositions of the form

$$\forall x \in [\boldsymbol{a}, \boldsymbol{b}]\colon \max(p,q)(\boldsymbol{x}) \;\Re\; K,$$

for any given m-variable polynomials p, q, constant value K, and real order relation $\Re \in \{\ge, >, \le, <\}$. If $\Re$ is one of $\ge$ or $>$, it suffices to check whether `omm.lbmin` $\Re\; K$. If $\Re$ is one of $\le$ or $<$, it suffices to check whether `omm.ubmax` $\Re\; K$. Furthermore, since an existential proposition of the form "$\exists \boldsymbol{c} \in [\boldsymbol{a}, \boldsymbol{b}]\colon \max(p,q)(\boldsymbol{c}) \ \Re \ K$" is equivalent to the negation of the universal proposition "$\forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]\colon \max(p,q)(\boldsymbol{x}) \ \neg\Re \ K$", the former type of existentially qualified propositions can also be checked using the algorithm `polyMinmax`. In this case, the witness $\boldsymbol{c}$ is either `omm.lbvar` or `omm.ubvar` depending on $\Re$.

5.4 Parameters `varsel`, `omm`, `localex`, and `globalex`

The parameter `varsel` is used to determine two things: (1) Which variable to subdivide at each recursive step, and (2) Whether to compute bounds to the left or the right first in that variable. The algorithm takes as inputs a representation `p` of an m-variable polynomial in Bernstein form p and a natural number `i`. It returns a pair $(\texttt{left}, \texttt{j})$, where `left` is a Boolean value and $\texttt{j} < m$. The value `left` being `true` means that the given variable should be subdivided to the left first, and `j` is a natural number representing the index of the variable to be subdivided. The most basic example of such a function is given by $\texttt{varsel}(\texttt{p}, \texttt{i}) = (\texttt{true}, \texttt{mod}(m, \texttt{i}))$, which alternates the variables and always computes range information on the left interval first. However, as noted in [16] and [20], there are much more efficient methods for choosing these variables and directions, including several based on derivatives. The function `varsel` is an input to the algorithm in PVS, so it can facilitate any subdivision scheme. One method that has been implemented in PVS is called `MaxVarMinDir`. This method chooses the variable for which the range between the first and last Bernstein coefficients, when all other variables are held constant, is greatest. In the algorithm `bernMinmax`, the function `varsel` is called on both `p` and `q`, and the answer that is used alternates between these answers.

The parameter `omm` is used for caching the current output of the algorithm. The function `between` is defined as follows

$$\begin{aligned} \texttt{between}(\texttt{omm}, \texttt{bmm}) \equiv \texttt{omm.lbmax} &\leq \texttt{bmm.lbmin} \ \text{ and} \\ \texttt{bmm.ubmax} &\leq \texttt{omm.ubmin}. \end{aligned}$$

It tests whether the output `bmm` at the current recursive step can contribute anything to the final output of the function once it is combined. At a given recursive step in the algorithm, if $\texttt{between}(\texttt{omm}, \texttt{bmm})$ returns `true`, then the output `bmm` of the current recursive step will not contribute to the overall output of the function since $\texttt{between}(\texttt{omm}, \texttt{bmm})$ implies that $\texttt{combine}(\texttt{omm}, \texttt{bmm}) = \texttt{omm}$.

The predicates `localex` and `globalex` are used to prune the executing tree depending on particular objectives. The predicate `localex` will be used to exit the algorithm locally and continue to the next recursive step. While both of these predicates are used in the algorithm to simply break recursion locally, the predicate `globalex` will be chosen so that if recursion breaks because `globalex` returns true, then every recursion above will also break, effectively resulting in a global exit from the algorithm. For instance, to check propositions of the form "$\forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]\colon \max(p,q)(\boldsymbol{x}) \geq K$", `localex` and `globalex` can be defined as follows.

$$\begin{aligned} \texttt{localex}(\texttt{omm}) &\equiv \texttt{omm.lbmin} \geq K, \\ \texttt{globalex}(\texttt{omm}) &\equiv \texttt{omm.lbmax} < K. \end{aligned}$$

In this case, if a box satisfies the inequality, there is no need to subdivide the box. On the other hand, if the negated inequality is satisfied by a point in the box, the algorithm must exit since a counterexample to the universally quantified inequality has been found.

6 Conflict Detection for Arbitrary Trajectories

It is noted in Section 3.3 that conflict detection between two trajectories P_o and P_i over a box $[\boldsymbol{a}, \boldsymbol{b}]$ can be reduced to determining whether the function $\mathtt{sqdist}_{D,H,P_o,P_i}$ ever takes a value less than 1 on $[\boldsymbol{a}, \boldsymbol{b}]$. The function $\mathtt{sqdist}_{D,H,P_o,P_i}$ is defined as the maximum of the functions $f(\boldsymbol{x}) = \frac{w_z^2}{H^2}$ and $g(\boldsymbol{x}) = \frac{w_x^2+w_y^2}{D^2}$, where $\mathbf{w} = P_o(\boldsymbol{x}) - P_i(\boldsymbol{x})$. The numerical constants D and H represent the minimum horizontal and vertical separation between the aircraft, respectively.

If P_o and P_i are polynomial functions, the functions f and g are also polynomials, and therefore the function $\mathtt{sqdist}_{D,H,P_o,P_i}$ is the maximum of two polynomials. The next proposition follows from Theorem 3.1 in Section 3.3 and the definition of the predicate *sound?*.

Proposition 9 *If sound?(*$\mathtt{sqdist}_{D,H,P_o,P_i}$*, omm,* $\boldsymbol{a}, \boldsymbol{b}$*) holds, then*

- *If omm.lbmin* ≥ 1*, then conflict?(*$\boldsymbol{a}, \boldsymbol{b}, P_o, P_i$*) does not hold.*
- *If omm.lbmax* < 1*, then conflict?(*$\boldsymbol{a}, \boldsymbol{b}, P_o, P_i$*) holds, and conflict is attained at the point lbvar, which is an element of* $[\boldsymbol{a}, \boldsymbol{b}]$*.*

According to Proposition 9 and Theorem 5.2 in Section 5, conflict between trajectories P_o and P_o can be determined by using the algorithm `polyMinmax` defined in Section 5.3. For this purpose, the local exit and global exit predicates used in the algorithm `bernMinmax` can defined as follows.

$$\begin{aligned}\mathtt{cd\_localexit(omm)} &\equiv \mathtt{omm.lbmin} \geq 1,\\ \mathtt{cd\_globalexit(omm)} &\equiv \mathtt{omm.lbmax} < 1.\end{aligned}$$

Alternatively, if the trajectories P_o and P_i are not defined by polynomials but are approximated by polynomial trajectories P_o' and P_i', the discussion in Section 3.3, including Proposition 1, implies that conflict information for P_o and P_i can be computed by considering the function $\mathtt{sqdist}_{D,H,P_o',P_i'}$, which is the maximum of two polynomials. In this case, the local exit and global exit predicates are parametric on D, H, ε_D, and ε_H, where ε_D and ε_H represent desired precisions in the horizontal and vertical dimensions as explained in Section 3.3. These parametric predicates are defined as follows.

$$\begin{aligned}\mathtt{cd\_localexit}_{D,H,\varepsilon_D,\varepsilon_H}\mathtt{(omm)} &\equiv \mathtt{omm.lbmin} \geq 1 + \delta_+(D, H, \varepsilon_D, \varepsilon_H),\\ \mathtt{cd\_globalexit}_{D,H,\varepsilon_D,\varepsilon_H}\mathtt{(omm)} &\equiv \mathtt{omm.lbmax} < 1 - \delta_-(D, H, \varepsilon_D, \varepsilon_H),\end{aligned}$$

where $\delta_+(D, H, \varepsilon_D, \varepsilon_H)$ and $\delta_-(D, H, \varepsilon_D, \varepsilon_H)$ are defined by Formula (8) in Section 3.3.

Based on these results, this section presents a formally verified conflict detection algorithm for arbitrary trajectories.

6.1 Verified Conflict Detection Algorithm

Suppose that P_o and P_i are polynomial trajectories from $\mathbb{R}^m$ into $\mathbb{R}^3$. The algorithm `cd_poly`, in Figure 9, can be used to check whether these two trajectories are in conflict on a given box $[\boldsymbol{a}, \boldsymbol{b}]$, i.e., whether *conflict?(*$D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i$*)* holds. The parametric function $\mathtt{cd\_poly}_{\mathtt{N},\varepsilon_D,\varepsilon_H}$ is a conflict detection algorithm as defined in Section 3.2. It has as inputs positive real numbers D, H, a box $[\boldsymbol{a}, \boldsymbol{b}]$, and polynomial trajectories P_o, P_i. It returns an element of type `CDOutcome` with the values `Unknown`, `NoConflict`,

```
$\texttt{cd\_poly}_{\texttt{N},\varepsilon_D,\varepsilon_H}(D, H, P_o, P_i, \boldsymbol{a}, \boldsymbol{b})$ : CDOutcome $\equiv$
    let  $p = \lambda \boldsymbol{x}\colon (P_o(\boldsymbol{x})_z - P_i(\boldsymbol{x})_z)^2/H^2$,
         $q = \lambda \boldsymbol{x}\colon (P_o(\boldsymbol{x})_{(x,y)} - P_i(\boldsymbol{x})_{(x,y)})^2/D^2$,
       omm $=$ polyMinmax($p, q, \boldsymbol{a}, \boldsymbol{b}$, N, MaxVarMinDir,
                 $\texttt{cd\_localexit}_{D,H,\varepsilon_D,\varepsilon_H}$,
                 $\texttt{cd\_globalexit}_{D,H,\varepsilon_D,\varepsilon_H}$) in
    if $\texttt{cd\_localexit}_{D,H,\varepsilon_D,\varepsilon_H}$(omm) then
      IsFalse
    elsif $\texttt{cd\_globalexit}_{D,H,\varepsilon_D,\varepsilon_H}$(omm) then
      Conflict(omm.lbvar)
    else
      Unknown
    endif
```

Figure 9: The function cd_poly

or LossAt($\boldsymbol{c}$), where $\boldsymbol{c} \in [\boldsymbol{a}, \boldsymbol{b}]$ is a point at which the trajectories loss separation. The parameter $\texttt{N} \in \mathbb{N}$ represents a maximum depth for the bounding algorithm polyMinmax presented in Section 5.3. The parameters ε_D and ε_H are usually 0, except when P_o and P_i are polynomial approximations of non-polynomial trajectories in which case they are small nonnegative real numbers.

Theorem 6.1 states that the algorithm $\texttt{cd\_poly}_{\texttt{N},\varepsilon_D,\varepsilon_H}$ is correct, as defined in Section 3.2, when $\varepsilon_D = \varepsilon_H = 0$.

Theorem 6.1 (Correctness for Polynomial Trajectories) *For all polynomial trajectories P_o and P_i from $\mathbb{R}^m$ into $\mathbb{R}^3$, boxes $[\boldsymbol{a}, \boldsymbol{b}] \in \mathbb{R}^m$, and depths $N \in \mathbb{N}$,*

1. *$\texttt{cd\_poly}_{N,0,0}(D, H, P_o, P_i, \boldsymbol{a}, \boldsymbol{b})$ = NoConflict implies*

$$\neg conflict?(D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i).$$

2. *$\texttt{cd\_poly}_{N,0,0}(D, H, P_o, P_i, \boldsymbol{a}, \boldsymbol{b})$ = LossAt($\boldsymbol{c}$) implies*

$$los?(D, H, P_o(\boldsymbol{c}), P_i(\boldsymbol{c}))$$

   and hence $\neg conflict?(D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i)$ holds.

Theorem 6.1 is a particular case of a more general theorem that takes the parameters ε_D and ε_H into account. Theorem 6.2 shows that the algorithm cd_poly can also be used to compute conflict information for arbitrary trajectories, assuming that polynomial approximations are known.

Theorem 6.2 (Correctness for Arbitrary Trajectories) *For all aircraft trajectories $P_o, P_i\colon \mathbb{R}^m \to \mathbb{R}^3$, polynomial trajectories $P'_o, P'_i\colon \mathbb{R}^m \to \mathbb{R}^3$, nonnegative real numbers $\varepsilon_D = \varepsilon_{oD} + \varepsilon_{iD}$ and $\varepsilon_H = \varepsilon_{oH} + \varepsilon_{iH}$, boxes $[\boldsymbol{a}, \boldsymbol{b}] \in \mathbb{R}^m$, and depths $N \in \mathbb{N}$, such that P'_o and P'_i are polynomial approximations of P_o and P_i, respectively, i.e., approx?($\boldsymbol{a}, \boldsymbol{b}, \varepsilon_{oD}, \varepsilon_{oH}, P_o, P'_o$) and approx?($\boldsymbol{a}, \boldsymbol{b}, \varepsilon_{iD}, \varepsilon_{iH}, P_i, P'_i$) hold,*

1. $\texttt{cd\_poly}_{N,\varepsilon_D,\varepsilon_H}(D, H, P'_o, P'_i, \boldsymbol{a}, \boldsymbol{b}) = \texttt{NoConflict}$ *implies*

$$\neg\textit{conflict?}(D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i).$$

2. $\texttt{cd\_poly}_{N,\varepsilon_D,\varepsilon_H}(D, H, P'_o, P'_i, \boldsymbol{a}, \boldsymbol{b}) = \texttt{LossAt}(\boldsymbol{c})$ *implies*

$$\textit{los?}(D, H, P_o(\boldsymbol{c}), P_i(\boldsymbol{c}))$$

and hence $\neg$*conflict?*$(D, H, \boldsymbol{a}, \boldsymbol{b}, P_o, P_i)$ *holds.*

Notice that the applications of cd_poly in Theorem 6.2 involve the polynomial trajectories P'_o and P'_i, whereas the conclusions involve P_o and P_i. Theorem 6.1 and Theorem 6.2 have both been formally proved in PVS. Theorem 6.1 follows trivially from Theorem 6.2 by setting $P_o = P'_o$, $P_i = P'_i$, $\varepsilon_D = 0$, and $\varepsilon_H = 0$. The proof of Theorem 6.2 follows from Theorem 5.2 in Section 5.3 and Proposition 1 in Section 3.3.

Theorem 6.1 and Theorem 6.2 are proved in PVS and they are available as part of the formal development at `http://shemesh.larc.nasa.gov/people/cam/Bernstein`.

6.2 An Example with Polynomial Trajectories

Consider the following two polynomial trajectories of type $\mathbb{R} \to \mathbb{R}^3$, meaning that they only depend on the time parameter and do not involve uncertainties.

$$\begin{aligned} P_o(t) = (&-3.2484 + 270.7\,t + 433.12\,t^2 - 324.83999\,t^3, \\ &15.1592 + 108.28\,t + 121.2736\,t^2 - 649.67999\,t^3, \\ &38980.8 + 5414.0\,t - 21656.0\,t^2 + 32484.0\,t^3) \\ P_i(t) = (&1.0828 - 135.35\,t + 234.9676\,t^2 + 3248.4\,t^3, \\ &18.40759 - 230.6364\,t - 121.2736\,t^2 - 649.67999\,t^3, \\ &40280.15999 - 10828.0\,t + 24061.9816\,t^2 - 32484.0\,t^3) \end{aligned}$$

These trajectories represent the 3D Euclidean positions of the ownship and intruder aircraft, respectively.

The unit of time for these trajectories is *hours* (hr), the unit of horizontal position is *nautical miles* (nm), and the unit of vertical position is *feet* (ft). The minimum separation standard is 5nm horizontally and 1000ft vertically, i.e., $D = 5$ and $H = 1000$. The lookahead time is assumed to be 3 minutes, i.e., $T = \frac{1}{20}$. Since the only variable of the polynomial is time, the box of interest for conflict detection is just the interval $[0, T]$.

At time $t = 0$ hours (current time), the positions of the ownship and intruder aircraft are $(-3.2484, 15.1592, 38980.8)$ and $(1.0828, 18.40759, 40280.15999)$, respectively. At this time, the aircraft are approximately 5.414 nm apart horizontally, and approximately 1299.36 ft apart vertically. Thus, given the minimum separation standard, the aircraft are not currently in loss of separation. However, the algorithm cd_poly predicts that the aircraft are in conflict. That is,

$$\texttt{cd\_poly}_{\texttt{N},0,0}(D, H, P_o, P_i) = \texttt{LossAt}(5105/262144), \tag{15}$$

where $\texttt{N} = 16$, which is the maximum recursion depth needed for the algorithm to terminate in this case. Theorem 6.1 states that these trajectories will be in loss of separation at time $t = \frac{5105}{262144}$, or in about 70 seconds. It follows that *conflict?*$(D, H, 0, T, P_o, P_i)$

holds. Indeed, at this time, the aircraft are approximately 4.999 `nm` apart horizontally and −999.92 `ft` vertically.

It is important to note that these results, which have been proved in PVS, are not subject to computation errors. For instance, the evaluation of `cd_poly` in Formula (15) is mathematically equal to `Conflict`(5105/262144). Furthermore, at time $t = \frac{5105}{262144}$ the horizontal distance between the aircraft is strictly less than 5 horizontally and strictly less than 1000 vertically.

6.3 Conflict Detection with Turning Trajectories

The trajectory of an aircraft in a steady turn, without considering uncertainties, is described by the parametric function $\mathtt{turn}_{\mathbf{s},R,\alpha,\omega,v_z} : \mathbb{R} \to \mathbb{R}$, with parameters $\mathbf{s}$, R, α, ω, and v_z, defined by Formula 3 in Section 3.1. Even though the function `turn` is not defined by polynomials, it can be approximated by polynomials. For any given natural number $n > 0$, the following polynomial trajectory approximates the trajectory $\mathtt{turn}_{\mathbf{s},R,\alpha,\omega,v_z}$.

$$\begin{aligned}\mathtt{turnpoly}_{\mathbf{s},R,\alpha,\omega,v_z,n}(t) = \mathbf{s} + (&\sum_{i=0}^{n} R\frac{(-1)^i}{(2i+1)!}(\alpha + t\,\omega)^{2i+1},\\ &\sum_{i=0}^{n} R\frac{(-1)^i}{(2i)!}(\alpha + t\,\omega)^{2i},\\ &t\,v_z).\end{aligned} \tag{16}$$

The summations in the first and second components of this trajectory represent Taylor series expansions of $R\sin(\alpha + t\cdot\omega)$ and $R\cos(\alpha + t\cdot\omega)$, respectively. The next lemma gives a bound for the error of this approximating trajectory, in terms of the predicate *approx?* defined in Section 3.3.

Lemma 6.1 *For any parameters $\mathbf{s}$, α, R, ω, v_z, and natural numbers n, the polynomial trajectory $\mathtt{turnpoly}_{\mathbf{s},\alpha,R,\omega,v_z,n}$ approximates $\mathtt{turn}_{\mathbf{s},\alpha,\omega,R,v_z}$ over the time interval between 0 and T, i.e.,*

$$approx?(0, T, \varepsilon_1 + \varepsilon_2, 0, \mathtt{turn}_{\mathbf{s},\alpha,\omega,R,v_z}, \mathtt{turnpoly}_{\mathbf{s},\alpha,R,\omega,v_z,n})$$

holds, where

$$\begin{aligned}\beta &\equiv \max(|\alpha|, |\alpha + T\cdot\omega|),\\ \varepsilon_1 &\equiv R\frac{\beta^{3+2n}}{(3+2n)!},\\ \varepsilon_2 &\equiv R\frac{\beta^{2+2n}}{(2+2n)!}.\end{aligned}$$

It is easy to see that ε_1 and ε_2 in Lemma 6.1 converge to zero as n approaches infinity. While this gives a bound for the error of the approximation, it depends on n. Thus, it is helpful to first specify the desired precision δ_D and then compute a number n such that $\varepsilon_1 + \varepsilon_2 < \delta_D$. This can be accomplished through a function named `exp_term_num` that takes as inputs a nonnegative real number x and a positive precision $\varepsilon > 0$. It returns a natural number $j > x + 1$ with the property that $\frac{x^k}{k!} < \varepsilon$ for all $k \geq 2\,j$. The existence of such a function follows from the fact that $\frac{x^k}{k!}$ converges to zero as k approaches infinity. There are many ways to define such a function, and in the PVS development it is defined recursively.

According to Lemma 6.1 and Theorem 6.2, the function `turnpoly` can be used to deduce conflict information for trajectories defined by `turn`. This is stated explicitly in the following theorems, which consider the case where both aircraft are turning and where only one aircraft is turning. These theorems have been formally proved in PVS.

Theorem 6.3 (Both Aircraft Are Turning) *For all P_o, P'_o, P_i, P'_i, $\mathbf{s}_o$, $\mathbf{s}_i$, α_o, α_i, ω_o, ω_i, R_o, R_i, v_{oz}, v_{iz}, β_o, β_i, n, `cd_outcome`, `N`, D, H, and T, if*

- $P_o = \mathtt{turn}_{\mathbf{s}_o,\alpha_o,\omega_o,R_o,v_{oz}}$,
- $P_i = \mathtt{turn}_{\mathbf{s}_i,\alpha_i,\omega_i,R_i,v_{iz}}$,
- $\beta_o = \max(|\alpha_o|, |\alpha_o + T \cdot \omega_o|)$,
- $\beta_i = \max(|\alpha_i|, |\alpha_i + T \cdot \omega_i|)$,
- $n = \mathtt{exp\_term\_num}(\varepsilon_D/(4 \max(R_o, R_i)), \max(\beta_o, \beta_i))$,
- $P'_o = \mathtt{turnpoly}_{\mathbf{s}_o,\alpha_o,R_o,\omega_o,v_{oz},n}$,
- $P'_i = \mathtt{turnpoly}_{\mathbf{s}_i,\alpha_i,R_i,\omega_i,v_{iz},n}$, *and*
- $\mathtt{cd\_outcome} = \mathtt{cd\_poly}(D, H, P'_o, P'_i, 0, T, \mathtt{N}, \varepsilon_D, 0)$,

then

1. $\mathtt{cd\_outcome} = \mathtt{NoConflict}$ *implies* $\neg$*conflict?*$(D, H, 0, T, P_o, P_i)$.
2. $\mathtt{cd\_outcome} = \mathtt{LossAt}(\mathtt{c})$ *and* $\varepsilon_D < D$ *implies conflict?*$(D, H, 0, T, P_o, P_i)$.

Theorem 6.4 (Only The Intruder Aircraft Is Turning) *For all P_o, P_i, P'_i, $\mathbf{s}_o$, $\mathbf{s}_i$, $\mathbf{v}_o$, α_i, ω_i, R_i, v_{iz}, β_i, n, `cd_outcome`, `N`, D, H, and T, if*

- $P_o = \mathtt{linear}_{\mathbf{s}_o,\mathbf{v}_o}$ *(cf. Section 3.1)*,
- $P_i = \mathtt{turn}_{\mathbf{s}_i,\alpha_i,\omega_i,R_i,v_{iz}}$,
- $\beta_i = \max(|\alpha_i|, |\alpha_i + T \cdot \omega_i|)$,
- $n = \mathtt{exp\_term\_num}(\varepsilon_D/(2\, R_i), \beta_i)$,
- $P'_i = \mathtt{turnpoly}_{\mathbf{s}_i,\alpha_i,R_i,\omega_i,v_{iz},n}$, *and*
- $\mathtt{cd\_outcome} = \mathtt{cd\_poly}(D, H, P_o, P'_i, 0, T, \mathtt{N}, \varepsilon_D, 0)$,

then

1. $\mathtt{cd\_outcome} = \mathtt{NoConflict}$ *implies* $\neg$*conflict?*$(D, H, 0, T, P_o, P_i)$.
2. $\mathtt{cd\_outcome} = \mathtt{LossAt}(\mathtt{c})$ *and* $\varepsilon_D < D$ *implies conflict?*$(D, H, 0, T, P_o, P_i)$.

7 Conclusion

A polynomial global optimization method for developing verifiable conflict detection algorithms for arbitrary trajectories has been proposed. The key idea of the method is to compute bounds for the maximum and minimum values of a function that is defined as the maximum of two polynomials. The bounds are computed using a recursive branch-and-bound algorithm via Bernstein polynomials.

The proposed method has been specified and verified in the interactive theorem prover PVS, and it assumes that computations are performed using real number semantics with infinite precision, a built-in feature of PVS. In a programming language, this semantics can be achieved using a library for arbitrary precision arithmetic such as GMP.[4]

In contrast to other global optimization methods, the method proposed here has been mechanically verified in a theorem prover. Therefore, it is guaranteed to be free of logical errors. Further, if implemented in a programming language, the correctness of the software only depends on the correctness of the library for arbitrary precision arithmetic, which is relatively small compared to general purpose global optimization environments such as GlobSol[5] and COCONUT[6]. The use of a specialized algorithm, which has been formally verified, as opposed to a general tool for global optimization could greatly simplify the certification process needed for the deployment of safety-critical CD&R systems.

Branch-and-bound methods for global optimization are usually exponential in the number of variables, and the method presented here is not the exception. However, recent developments in branching and pruning heuristics and efficient representations of multivariate polynomials make Bernstein polynomial methods practical in many cases. Future research will study the feasibility of the proposed approach for airborne conflict detection, where the computational resources are scarce and the frequency of execution is on the order of 1Hz.

References

[1] Antonio Bicchi and Lucia Pallottino. On optimal cooperative conflict resolution for air traffic management systems. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):221–231, December 2000.

[2] Heinz Erzberger and Karen Heere. Algorithm and operational concept for resolving short-range conflicts. In *Proceedings of the 26th International Congress of the Aeronautical Sciences, ICAS 2008*, Anchorage, Alaska, USA, September 2008.

[3] Rida T. Farouki. The Bernstein polynomial basis: a centennial retrospective. *Computer Aided Geometric Design*, 29:379–419, 2012.

[4] Jürgen Garloff. The Bernstein algorithm. *Interval Computations*, 4:154–168, 1993.

[5] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[6] George Hagen, Ricky Butler, and Jeffrey Maddalon. Stratway: A modular approach to strategic conflict resolution. In *Preceedings of 11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Virgina Beach, VA, September 2011.

[7] David A. Karr and Robert A. Vivona. Conflict detection using variable four-dimensional uncertainty bounds to control missed alerts. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, number AIAA 2006-6057, Keystone, Colorado, USA, August 2006.

[4] `http://gmplib.org/`.

[5] `http://interval.louisiana.edu/GlobSol`.

[6] `http://www.mat.univie.ac.at/~coconut/coconut-environment`.

[8] James Kuchar and Lee Yang. A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):179–189, December 2000.

[9] G. G. Lorentz. *Bernstein Polynomials.* Chelsea Publishing Company, New York, N.Y., second edition, 1986.

[10] Jeffrey Maddalon, Ricky Butler, César Muñoz, and Gilles Dowek. Mathematical basis for the safety analysis of conflict prevention algorithms. Technical Memorandum NASA/TM-2009-215768, NASA, Langley Research Center, Hampton VA 23681-2199, USA, June 2009.

[11] César Muñoz and Anthony Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 2012.

[12] Anthony Narkawicz and César Muñoz. Time of closest approach in three-dimensional airspace. Technical Memorandum NASA/TM-2010-216857, NASA, Langley Research Center, Hampton VA 23681-2199, USA, October 2010.

[13] Anthony Narkawicz, César Muñoz, and Gilles Dowek. Provably correct conflict prevention bands algorithms. *Science of Computer Programming*, 2011. In Press.

[14] Michael S. Nolan. *Fundamentals of Air Traffic Control.* Brooks Cole, 4 edition, July 2003.

[15] Sam Owre, John Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proceeding of the 11th International Conference on Automated Deductioncade*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer, June 1992.

[16] M. Arounassalame P. S. V. Nataraj. A new subdivision algorithm for the Bernstein polynomial approach to global optimization. *International Journal of Automation and Computing*, 4(4):342, 2007.

[17] Russell Paielli. Modeling manuever dynamics in air traffic conflict resolution. *Journal of Guidance, Control, and Dynamics*, 26(3):407–215, May–June 2003.

[18] Lucia Pallottino, Eric Feron, and Antonio Bicchi. Conflict resolution problems for air traffic management systems solved with mixed integer programming. *IEEE Transactions on Intelligent Transportation Systems*, 3(1):3–11, March 2002.

[19] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods, 16th International Symposium on Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 547–562. Springer, 2009.

[20] Shashwati Ray and P. S. Nataraj. An efficient algorithm for range computation of polynomials using the Bernstein form. *J. of Global Optimization*, 45:403–426, November 2009.

[21] Arthur Richards and Jonathan How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference*, volume 3, pages 1936–1941, 2002.

[22] Walter Rudin. *Principles of Mathematical Analysis.* McGraw-Hill, third edition, 1976.

[23] Andrew Paul Smith. Fast construction of constant bound functions for sparse polynomials. *J. of Global Optimization*, 43:445–458, March 2009.

[24] Harry Swenson, Richard Barhydt, and Michael Landis. Next Generation Air Transportation System (NGATS) Air Traffic Management (ATM)-Airspace Project, June 2006.